Group 3

Arranged by

Ahmed Hosam
Abdelrahman Hanafy
Ahmed Ramadan
Farah Ahmed Anany
Khloud Maged
Mahmoud Awad

Prepared for

Prof. Cherine Fathi
Eng. Marwan Nour
Eng. Yahya Emad
Eng. Ahmed Noaman

Voice Mail System Report

CC431 Computer Networks

Contents

Express

HI !! 👋 👋

Part One

project description

The project involves the implementation of a client-server voice mail application that enables users to send voice messages to each other. This application will provide a convenient and efficient way for users to communicate using recorded voice instead of traditional text-based messages. Users will be able to record voice mails, send them to other users, and receive voice mails from others within the application.

Technology Stack

The client-server voice mail application utilizes a powerful technology stack. Node.js and Express form the server foundation, providing scalability and efficient routing. Real-time communication is facilitated by Socket.io, ensuring instant message transmission. PostgreSQL manages voice mail data securely. JSON Web Tokens (JWT) enable secure user authentication. On the client side, React.js creates a dynamic user interface, while React-Mic adds audio recording capabilities. Socket.io establishes a seamless connection, enabling real-time updates. This combination results in a feature-rich, secure voicemail application that offers a seamless messaging experience.

By leveraging Node.js, Express, Socket.io, PostgreSQL, and JWT on the server, and React.js, React-Mic, and Socket.io on the client, the voice mail application provides reliability, scalability, and responsiveness. Users can enjoy exchanging voice messages in real-time through an intuitive interface, benefiting from the application's robust technology stack.

Socket Events breakdown

Part Two

userConnected

The event data being emitted is an object containing the property "userId" with a value that was previously defined.

Here's a breakdown of what the code does:

- socket.emit: This is a function provided by the Socket.IO library that allows the server to send events to the connected clients. In this case, it is being used to emit the "userConnected" event.
- "userConnected": This is the name of the event being emitted. It can be any custom name you choose.
- { userId }: This is an object being passed as the data payload along with the event. It contains a property called "userId" with a value that was previously defined.
- The event is emitted specifically to the client connected to the socket that triggered this code. Other connected clients will not receive this event unless explicitly emitted to them.

Overall, this code is emitting the "userConnected" event to the client connected to the socket, providing information about the user's ID. The client can listen for this event and handle it accordingly.

sendMail

It performs the necessary logic to send an email and notify the relevant client about the status of the email. Here's a breakdown of what the code does:

- socket.on: This is a function provided by Socket.IO to handle incoming events from the client. In this case, it listens for the "sendMail" event.
- "sendMail": This is the name of the event being handled. It should match the event name emitted from the client.
- async (data) => { ... }: This is an asynchronous function that will be executed when the "sendMail" event is received. It takes the data payload sent from the client as an argument.
- The code then performs various actions related to sending the email, such as saving the email to a database, retrieving updated mail lists, and determining the recipient's socket ID.
- io.to(receiverSocketId).emit("mailReceived", { mails_list: mails_list });: This emits a "mailReceived" event to the client with the receiver's socket ID. It sends the updated mail list as the payload.
- socket.emit("mailSent", { message: "Mail sent successfully" });: This emits a "mailSent" event back to the client that triggered the "sendMail" event. It sends a success message as the payload.

Overall, this code handles the "sendMail" event, performs necessary logic for sending an email, notifies the recipient client about the received mail, and sends a response back to the client that sent the original "sendMail" event.

getMails

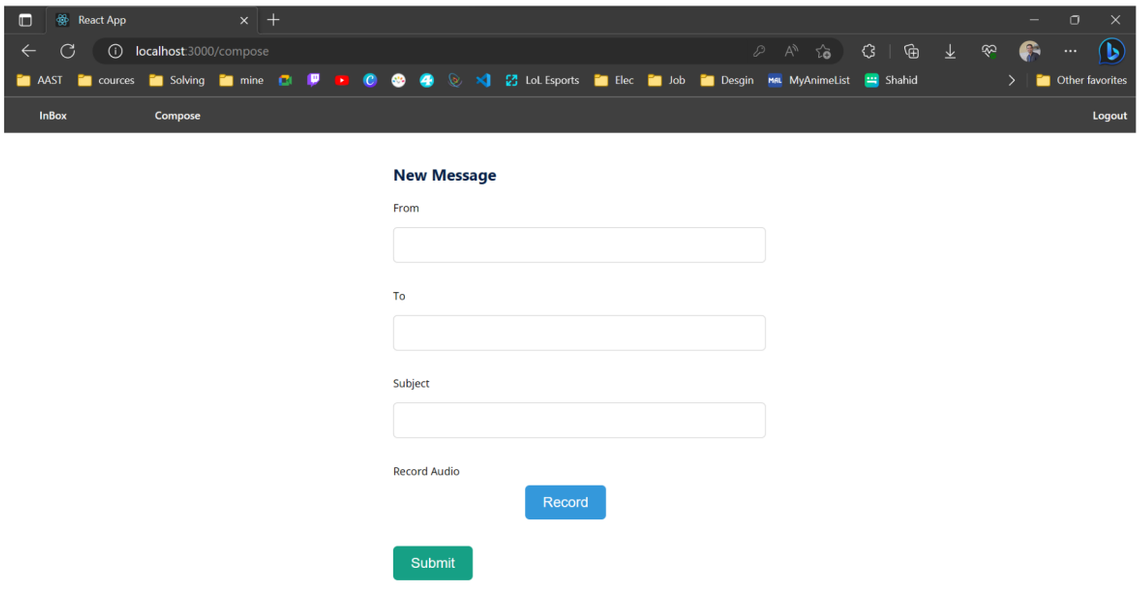
It retrieves the emails associated with a user and sends them back to the client. Here's a breakdown of what the code does:

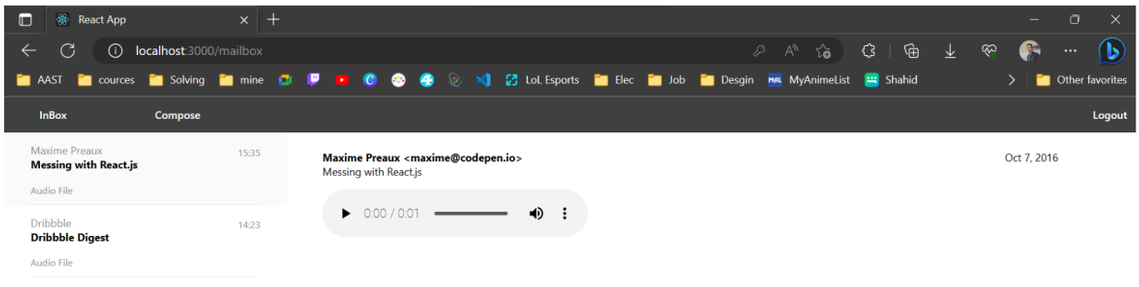
- socket.on: This is a function provided by Socket.IO to handle incoming events from the client. In this case, it listens for the "getMails" event.
- "getMails": This is the name of the event being handled. It should match the event name emitted from the client.
- async () => { ... }: This is an asynchronous function that will be executed when the "getMails" event is received. It does not take any arguments.
- The code then performs actions to retrieve the emails associated with the user. It calls the getMailFromUserId function to get the user's email using the userId variable. Then, it calls the getMailsByReceiverEmail function to retrieve the list of mails associated with that email.
- socket.emit("mailReceived", { mails_list: mails_list });: This emits a "mailReceived" event back to the client that triggered the "getMails" event. It sends the retrieved mail list as the payload.

Overall, this code handles the "getMails" event, retrieves the emails associated with the user, and sends them back to the client through the "mailReceived" event.

Application Screenshots

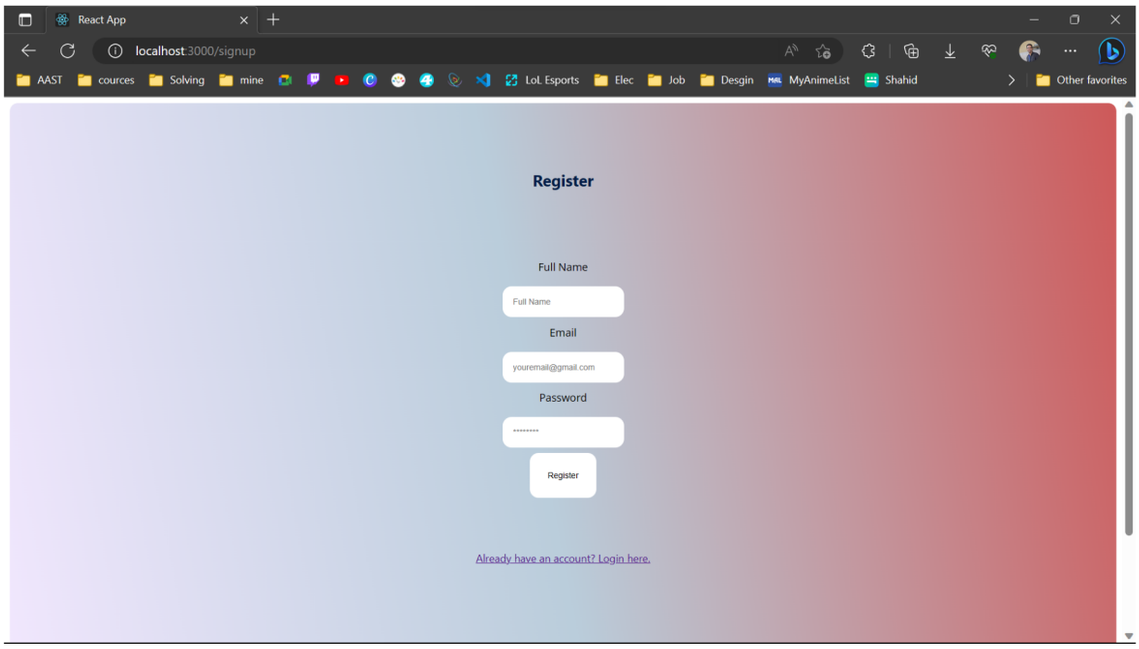
Part Three

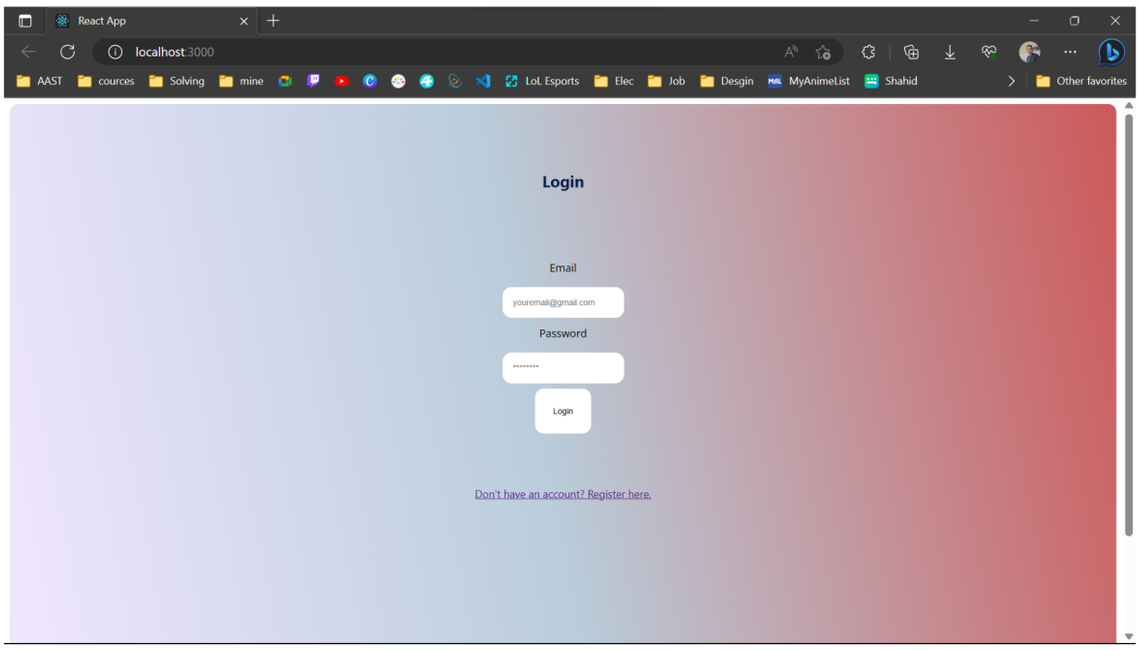




Application Screenshots

Part Three





Task Division

Part Four

01 ST

Project planning

All Members

Backend (Server)

Ahmed Hossam
Khloud Maged

02 ND

03 RD

Frontend

Farah Anany
Ahmed Ramadan
Mahmoud Awad

Client Integration

Abdelrahman Hanafy
Ahmed Ramadan

04 TH

05 TH

Documentation

Abdelrahman Hanafy
Farah Anany